Moving beyond prototypes: Building resilience at scale in your IoT application

Alina Dima

HighLoad++ Armenia

Co-organizer

Yandex

Agenda

About me

Introduction to IoT and MQTT

Why is resilience in IoT applications important?

Maturity Model for resilient IoT applications

Key Takeaways and Q&A

About me

Alina Dima

- Senior Developer Advocate, AWS IoT Global service team.
- Love problem solving.
- Close to 20 years engineering experience, designing and building Serverless and IoT solutions at scale.

Goals of the session

Understand some of the challenges that lead to data loss, data inaccuracy, or data delays in IoT applications.

Help you make informed decisions on how to mitigate the challenges.

Identify the tools to mitigate the challenges.

Build a maturity model.

- We **will** discuss resilience as a crucial factor for data consistency in IoT applications.

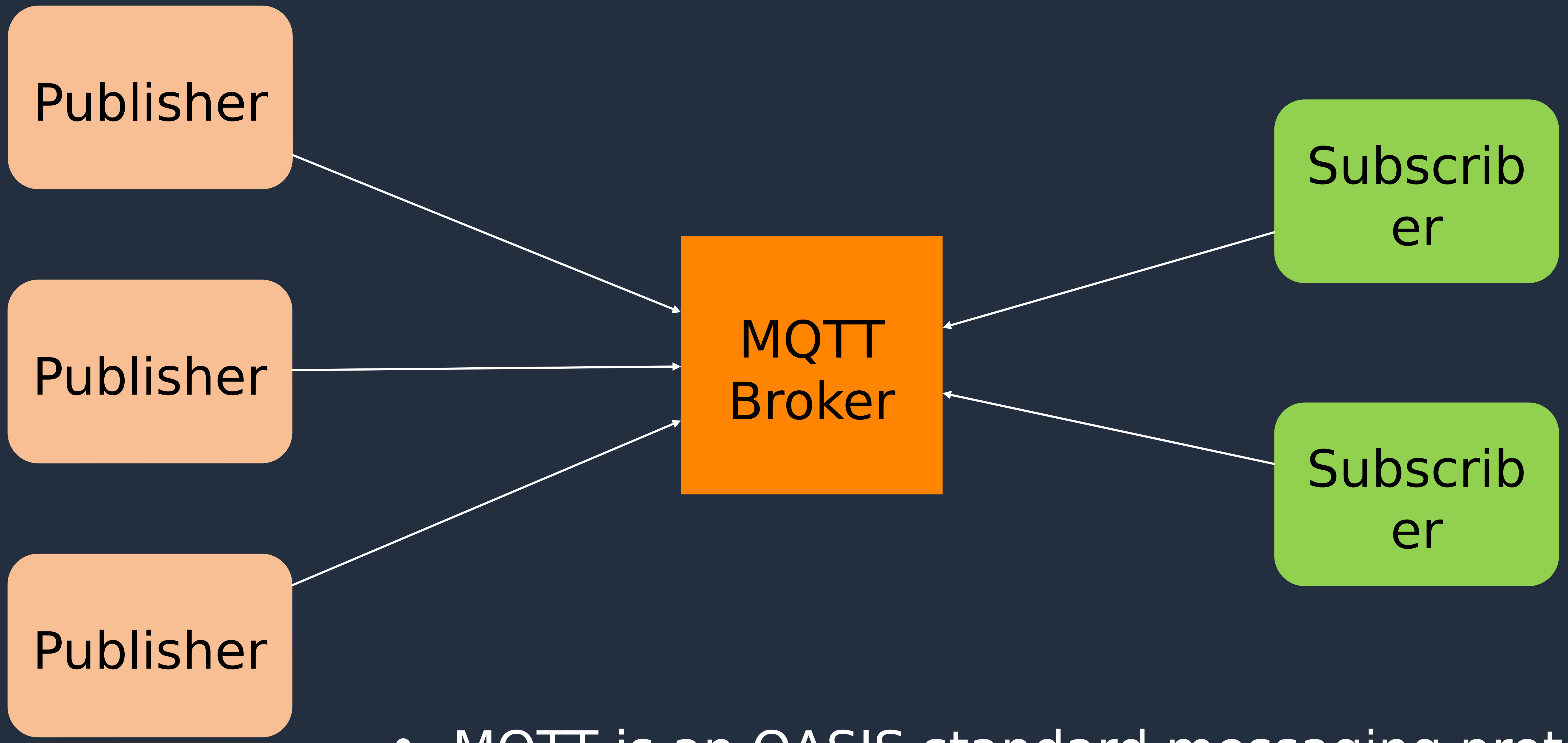
- This session is **not** a comparison between tools.
- It is not a comparison between AWS services.

Introduction to IoT and MQTT

What is IoT?

- Billions of connected devices and generating data and actuating.
- Data ***can*** become an organizational asset. But only if it is reliable.
- It is not possible to have reliable data with a non-resilient IoT application.

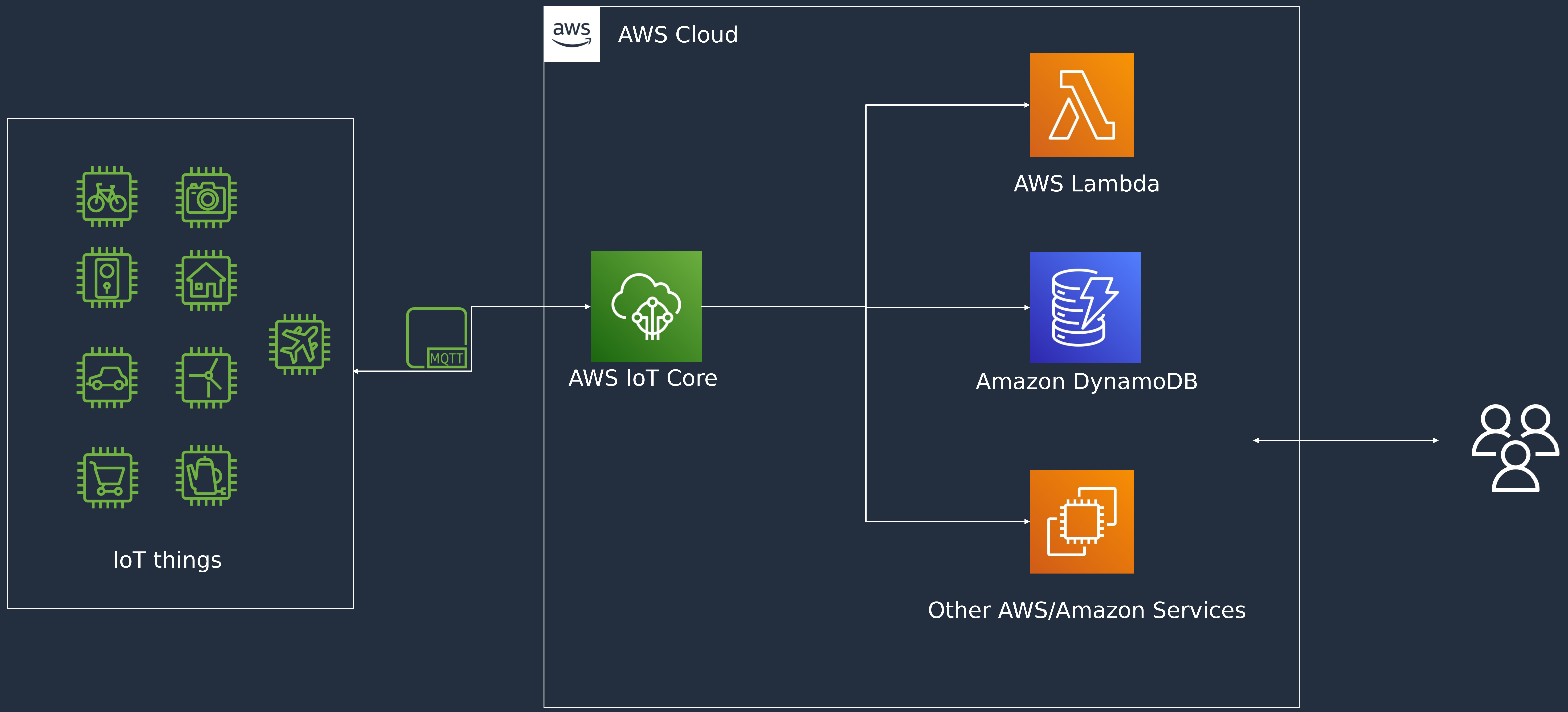
Introduction to MQTT



- Bi-directional communication.
- Decoupled producers and consumers.
- Catered for unreliable connectivity scenarios.

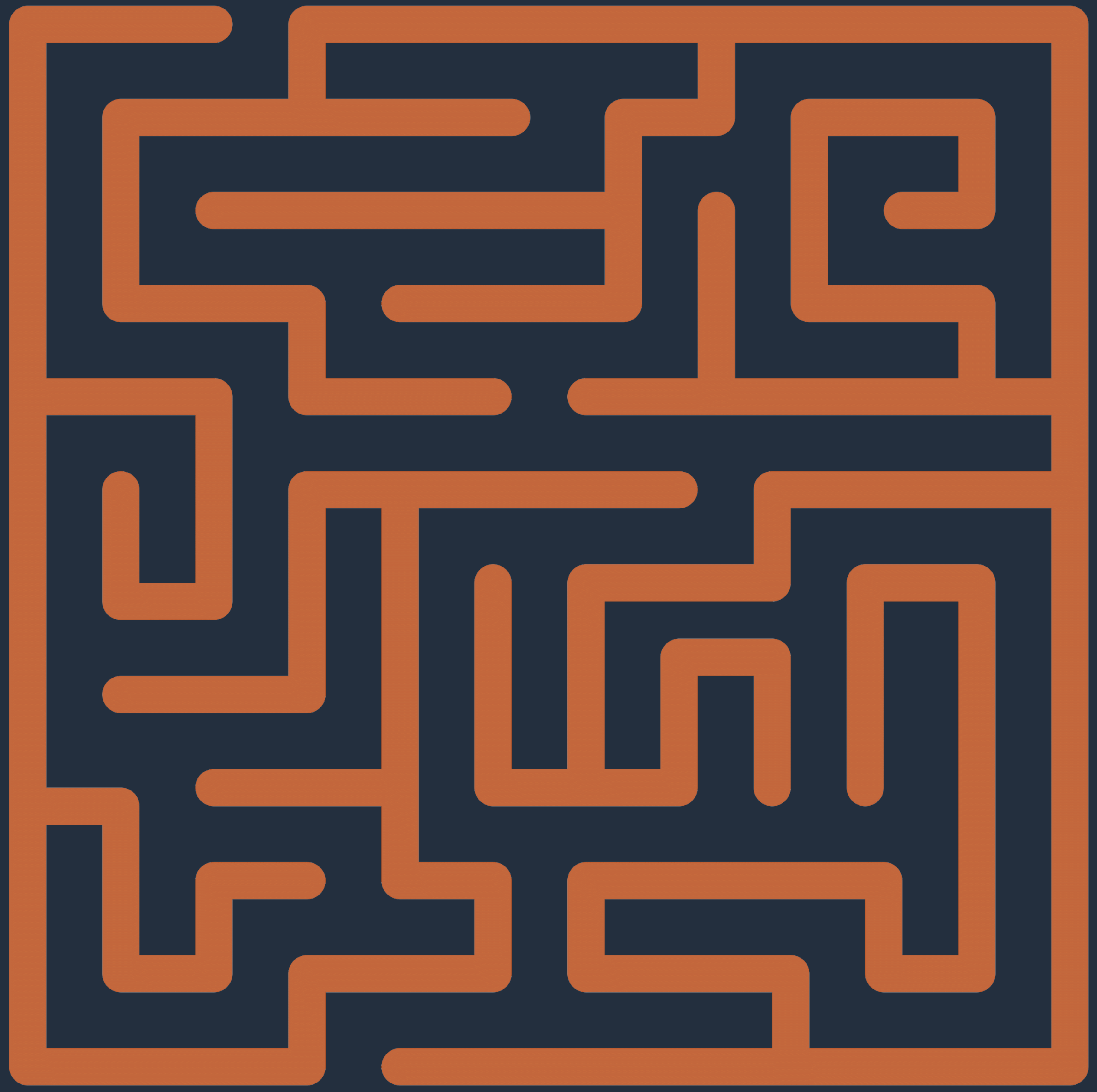
- MQTT is an OASIS standard messaging protocol for the Internet of Things (IoT).
- Application layer, typically over TCP/IP.

AWS IoT


AWS Cloud
IoT things
MQTT
AWS IoT Core
AWS Lambda
Amazon DynamoDB
Other AWS/Amazon Services

Challenges you might face when building IoT

- Complexity and constraints.
- Getting started and moving past prototypes.
- Problems are hard to identify early, let alone fix.
- Shared responsibility model - IoT is a means to an end.
    - Not just Cloud versus your Application.
    - But also between your different internal teams.



Why do we care about resilience in IoT applications ?

Let's look at a real world scenario

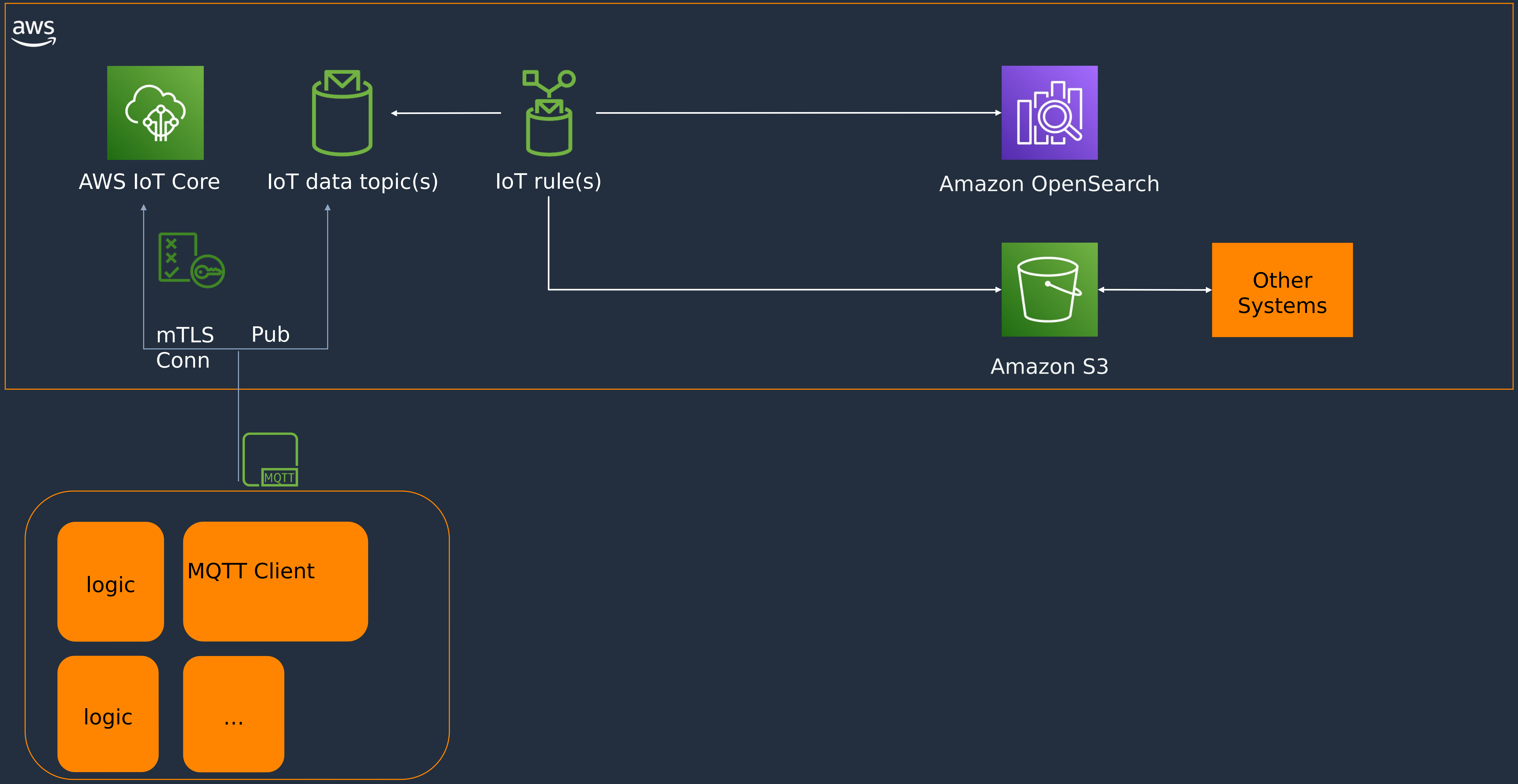
Let's look at a real world scenario

- Monitor elevator activity.
- Read and understand elevator data:
  - Door openings/closings and speeds.
  - Speed of movement.
  - How long does a run take?
- Predict failures and auto-schedule maintenance windows.

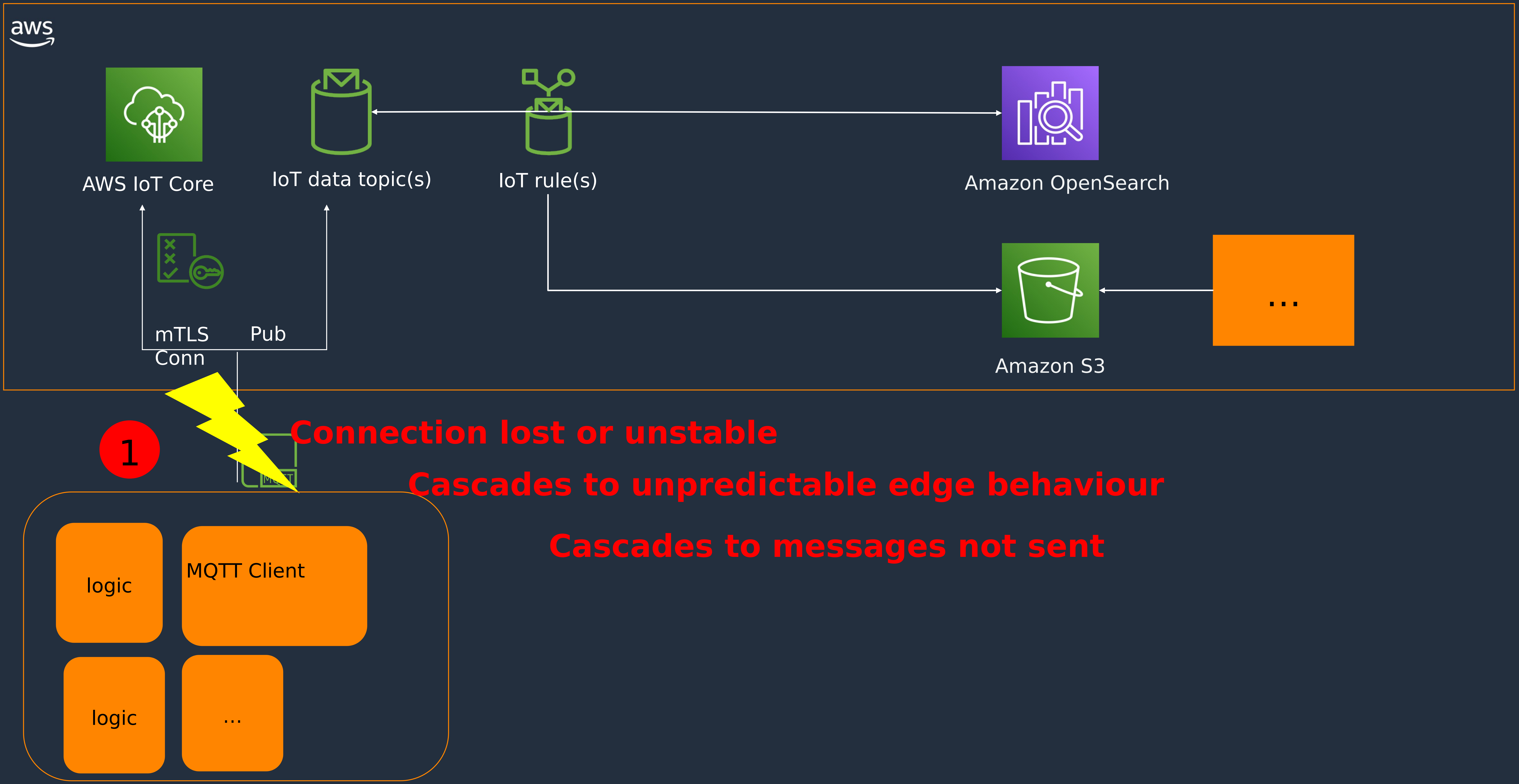
BUT, we have a problem with the existing solution…

- On production, the solution does not meet the requirements:
  - Devices also go offline and stay offline or fluctuate between offline and online.
  - Data is missing: real time events and historical data gaps.
  - Data has been flagged as unreliable by the analytics team.
  - Engineers across different teams cannot agree where the problem lies.

High Level Architecture



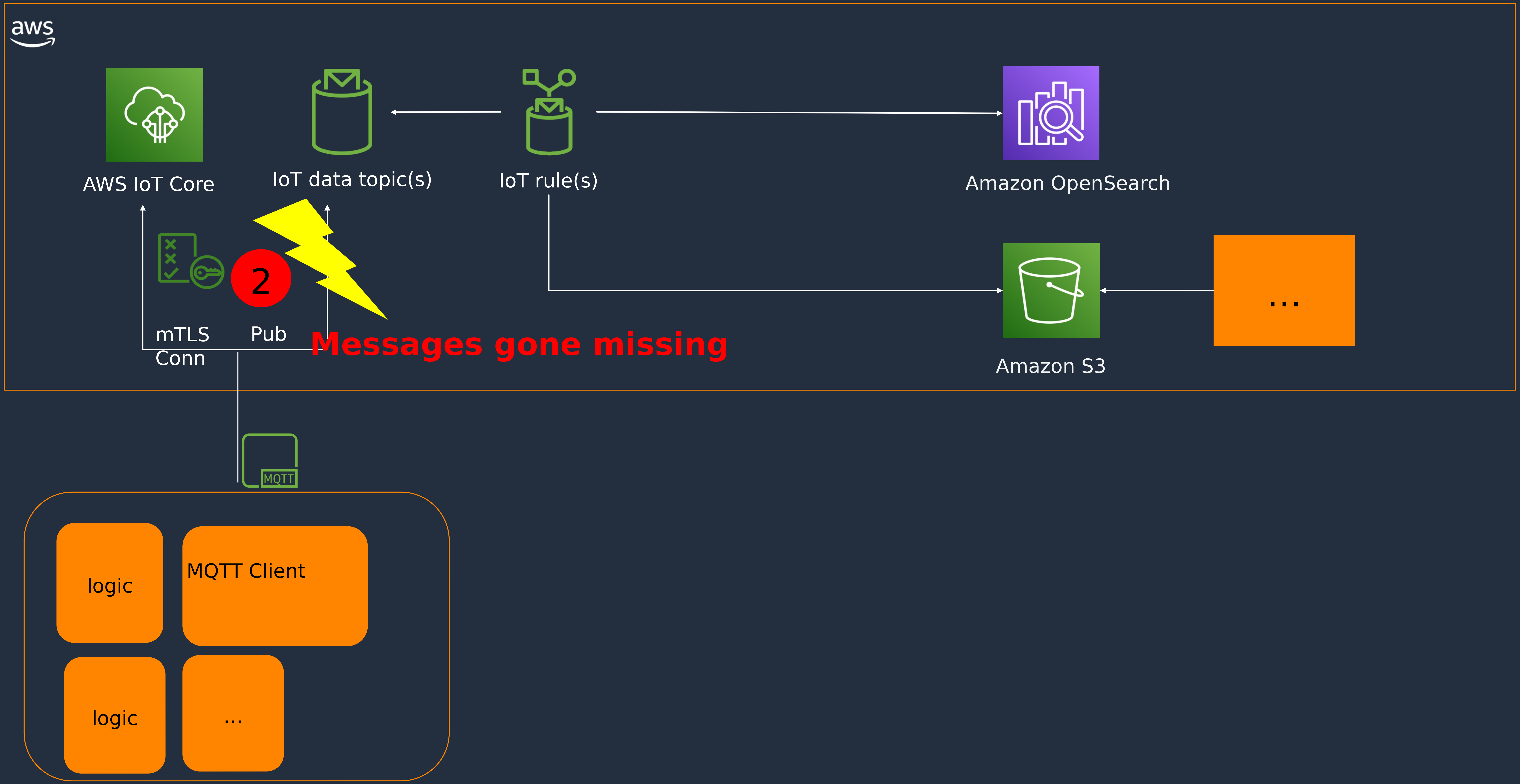
What can go wrong?



Connection lost or unstable

Cascades to unpredictable edge behaviour

Cascades to messages not sent



AWS IoT Core

IoT data topic(s)

IoT rule(s)

Amazon OpenSearch

mTLS
Conn

2

Pub

Messages gone missing

Amazon S3

...

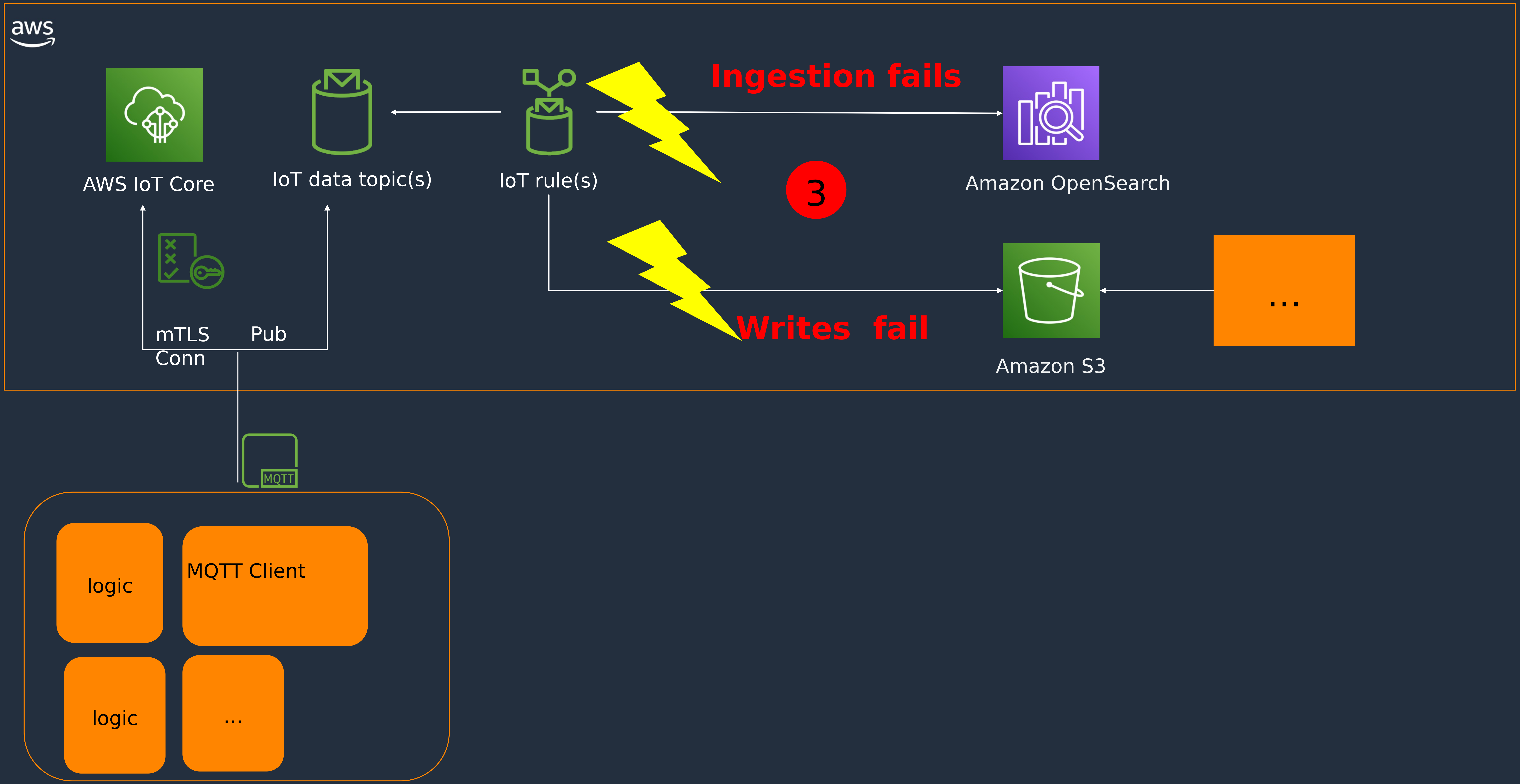
MQTT

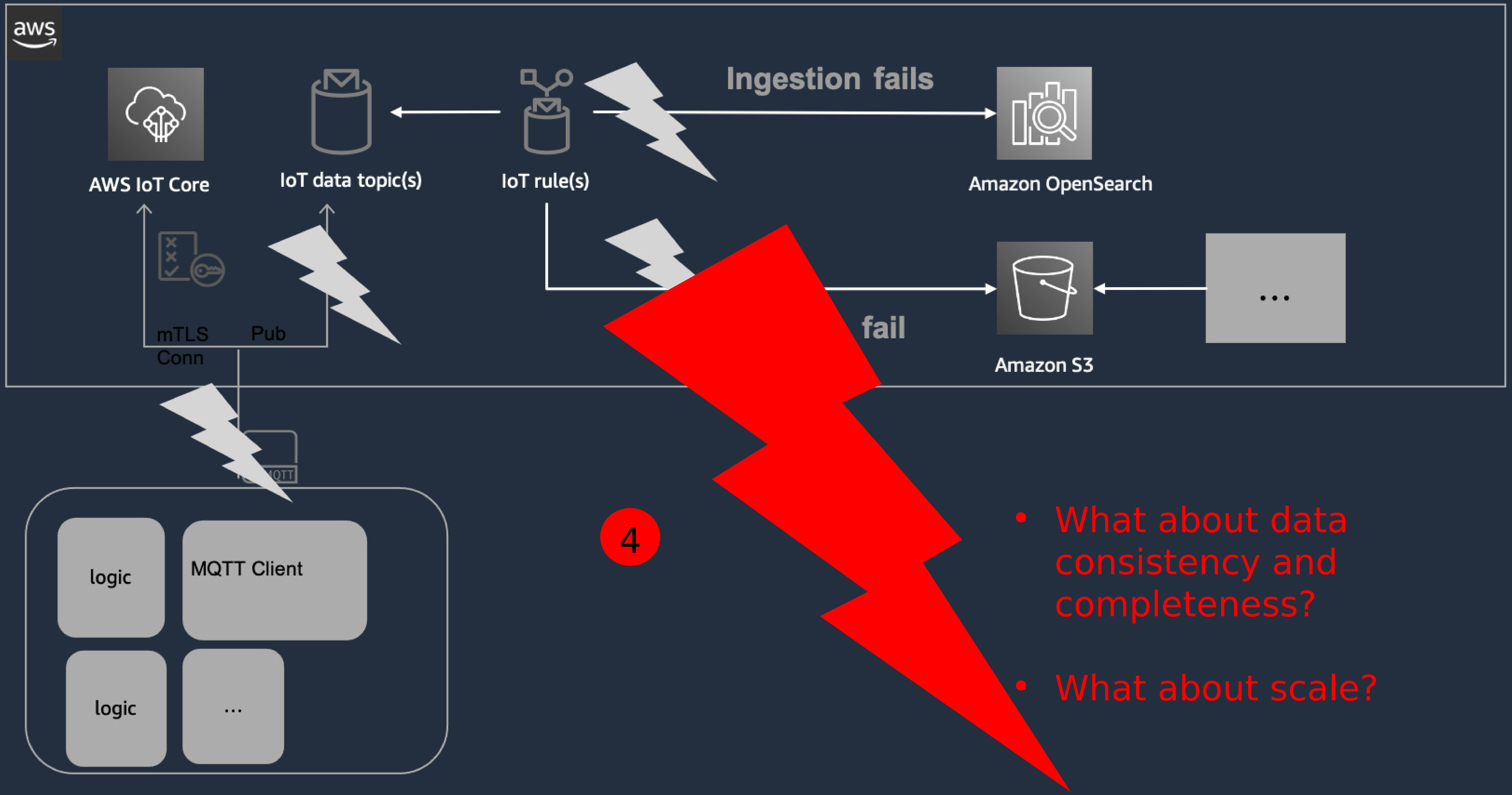
logic

MQTT Client

logic

...


aws
AWS IoT Core
IoT data topic(s)
IoT rule(s)
Ingestion fails
Amazon OpenSearch
3
mTLS
Conn
Pub
Writes fail
Amazon S3
...
MQTT
logic
MQTT Client
logic
...



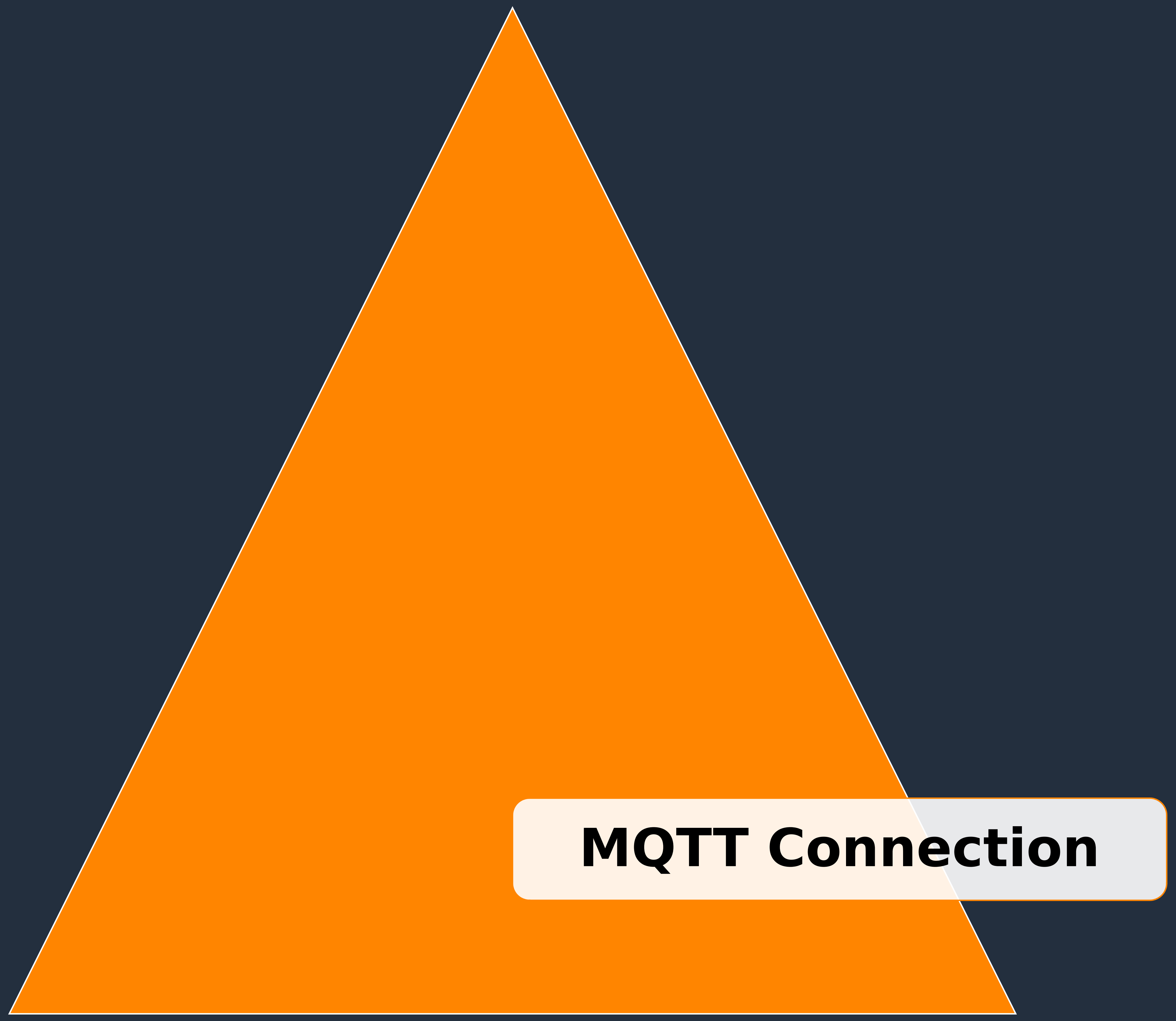
- What about data consistency and completeness?
- What about scale?

How do we solve these issues?

How do we solve **reliability** issues?

By building resilience!

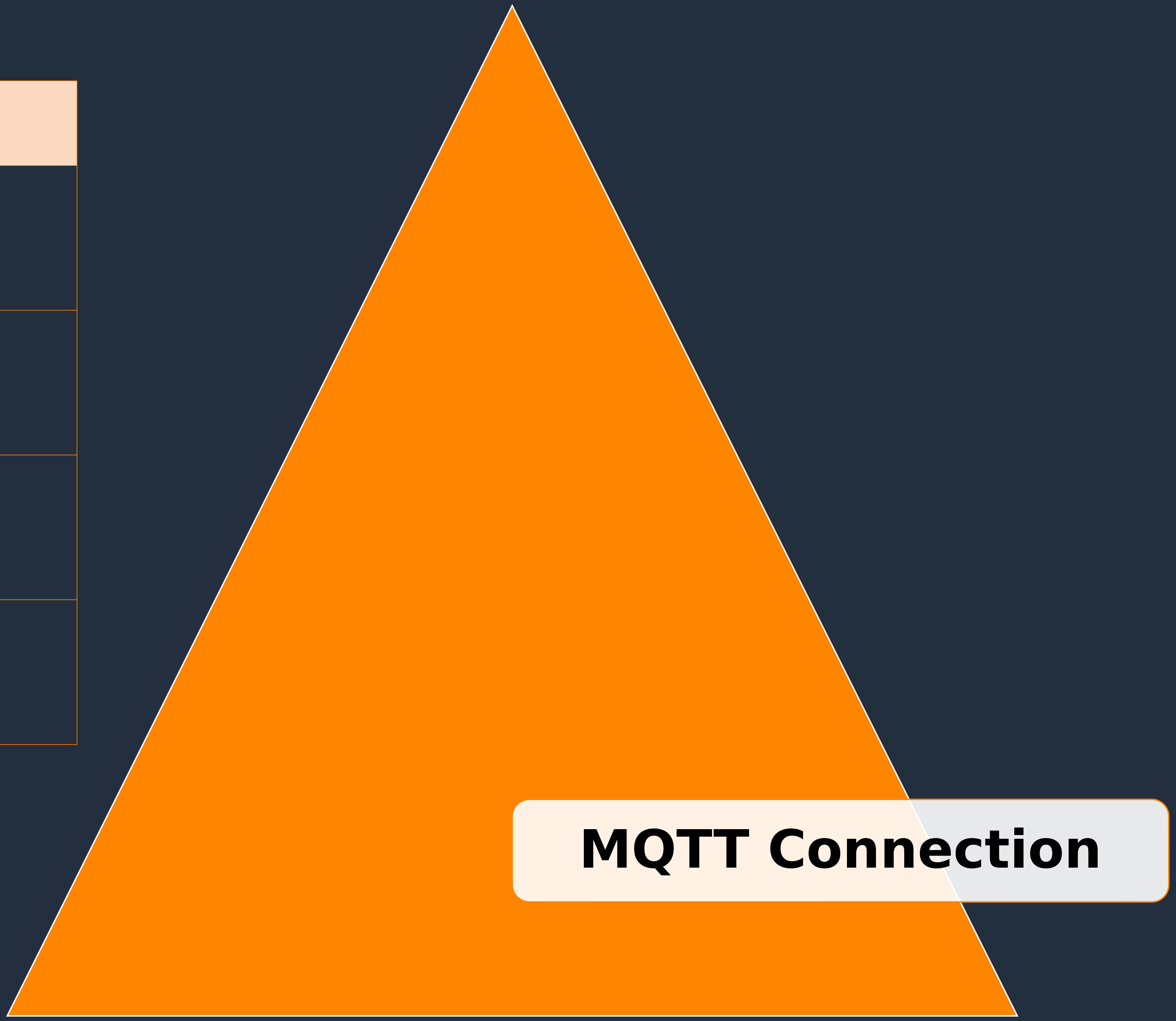
Resilient MQTT Connection



MQTT Connection

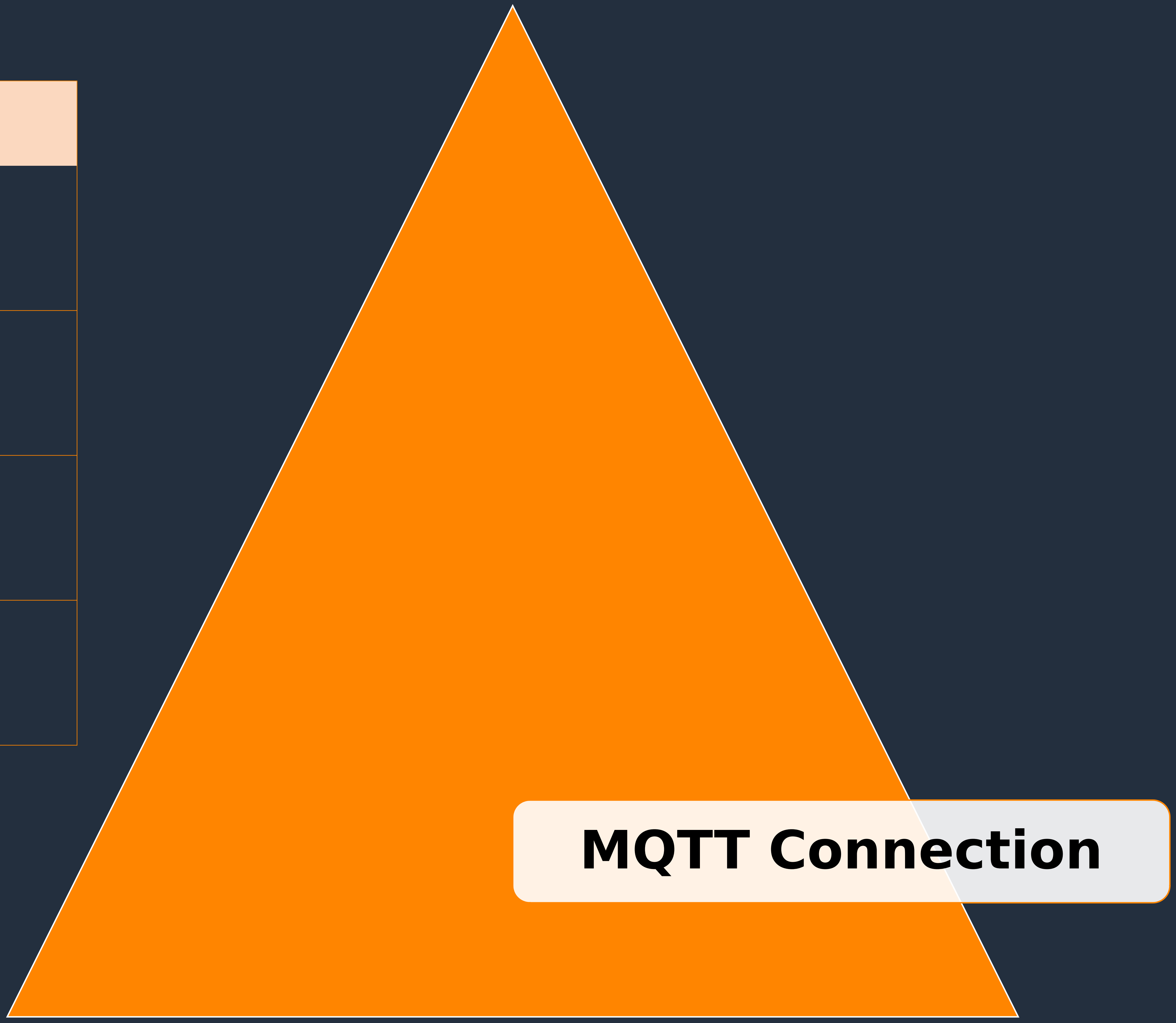
Resilient MQTT Connection

MQTT Protocol
Keep Alive /Heartbeat.
Client Takeover.
Last Will Testament (LWT).
Persistent Sessions.



Resilient MQTT Connection

MQTT Protocol
Keep Alive /Heartbeat.
Client Takeover.
Last Will Testament (LWT).
Persistent Sessions.



Application Level
Understand your MQTT client library.
Manage the MQTT connection lifecycle.
Listen for/handle MQTT events, connects, interrupts, resumes etc.
Connection State Checks.
Track and recover from connection errors (client and server-side).
Automatic reconnects, with jitter and/or exponential backoff strategy.
Design devices to have an accurate time.
Use tools that allow you to test your MQTT implementation, like AWS IoT Device Advisor.

Resilient Message Delivery

Message Delivery
MQTT Connection

Resilient Message Delivery

MQTT Protocol
Quality of Service. – QoS 1 for reliable message transmission.
Message Queueing.
Retained Messages.
Persistent Sessions.

Message Delivery
MQTT Connection

Resilient Message Delivery

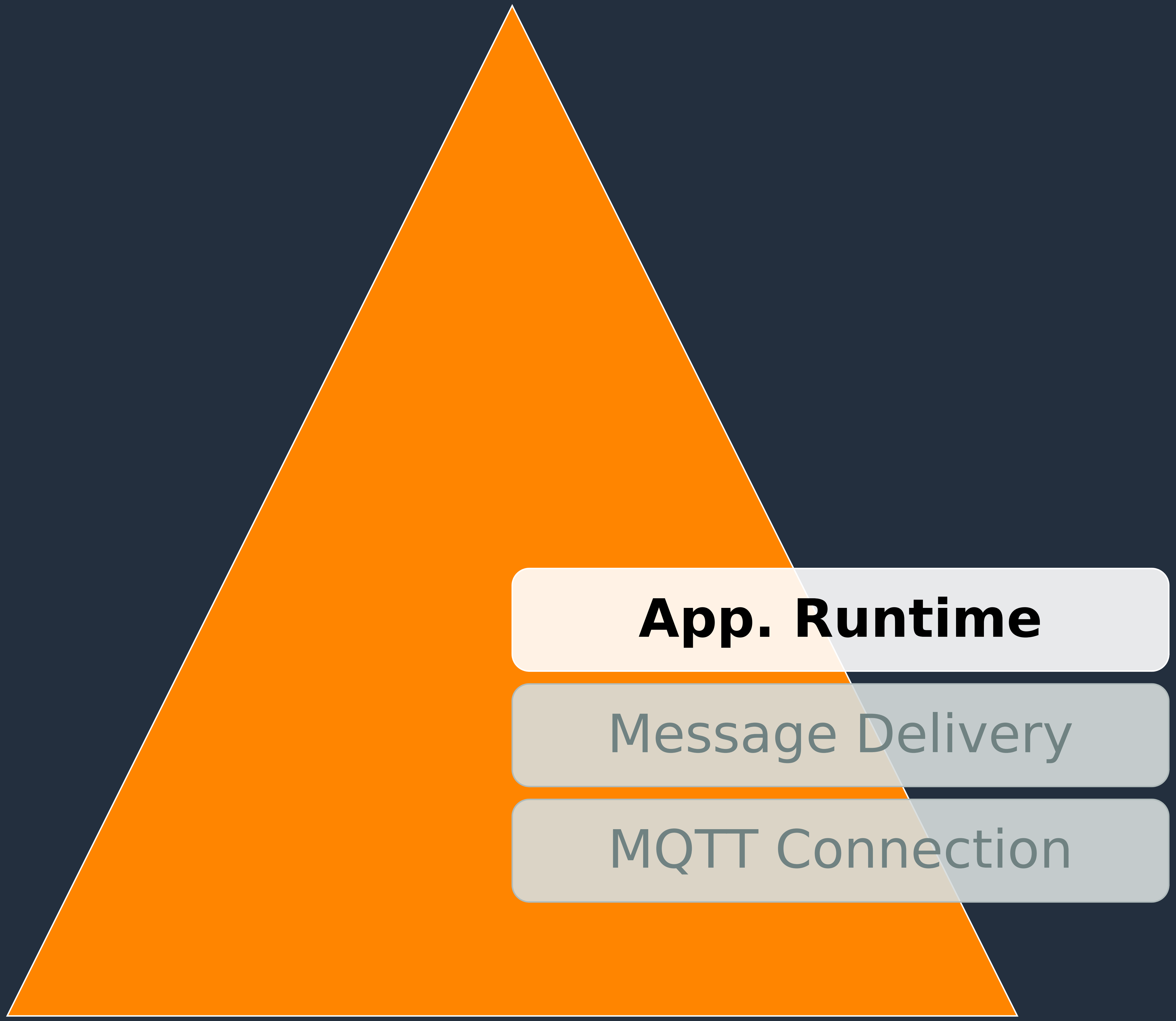
MQTT Protocol
Quality of Service. – QoS 1 for reliable message transmission.
Message Queueing.
Retained Messages.
Persistent Sessions.

Message Delivery

MQTT Connection

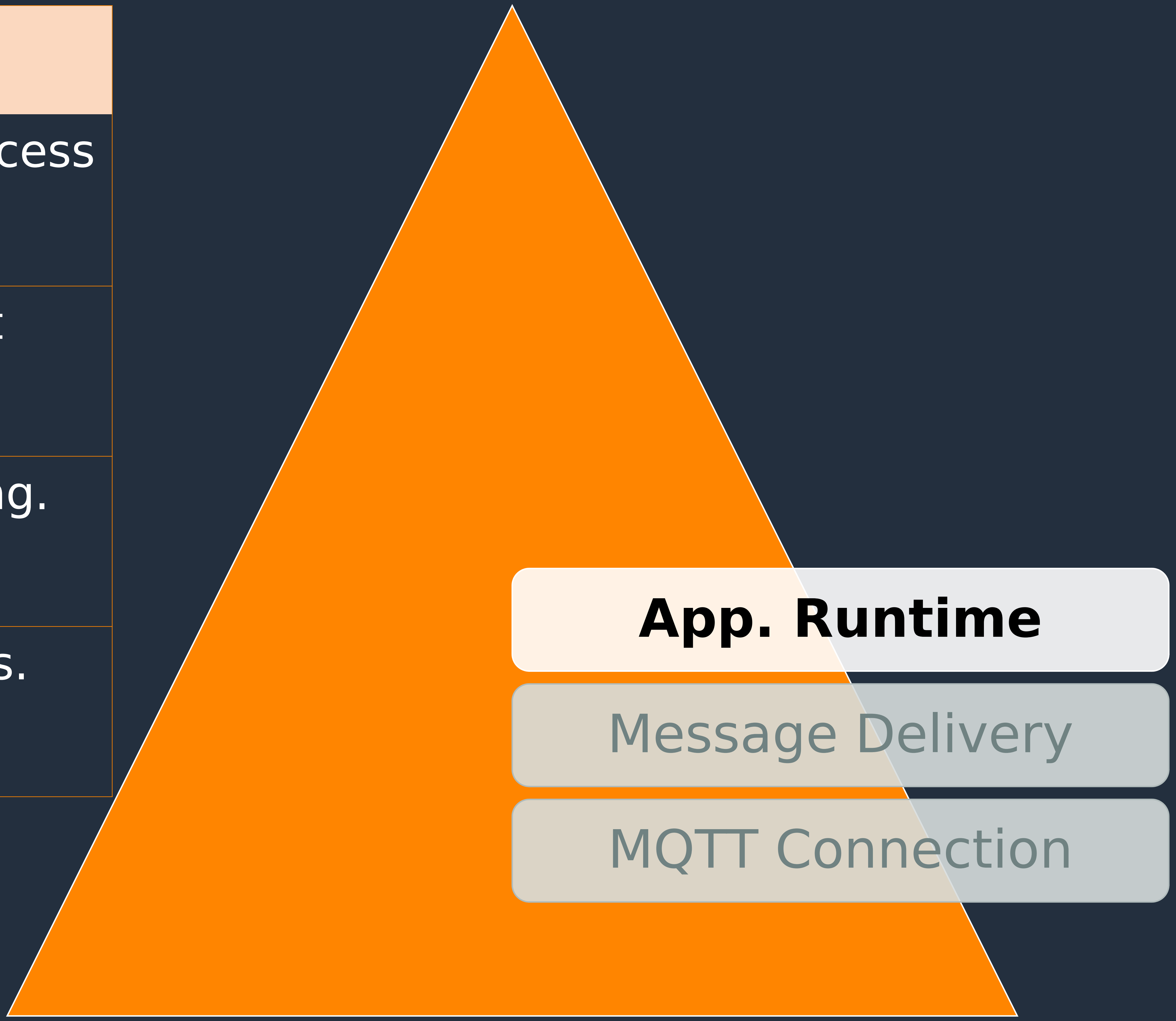
Application Level
Encapsulate the MQTT transport layer.
MQTT message buffering for short time connection loss.
Track success/failure of message delivery – PUBACKs.
Mitigate failed delivery.
Offline data storage strategy.
Optimize data sent from devices to backend services.

Runtime resilience

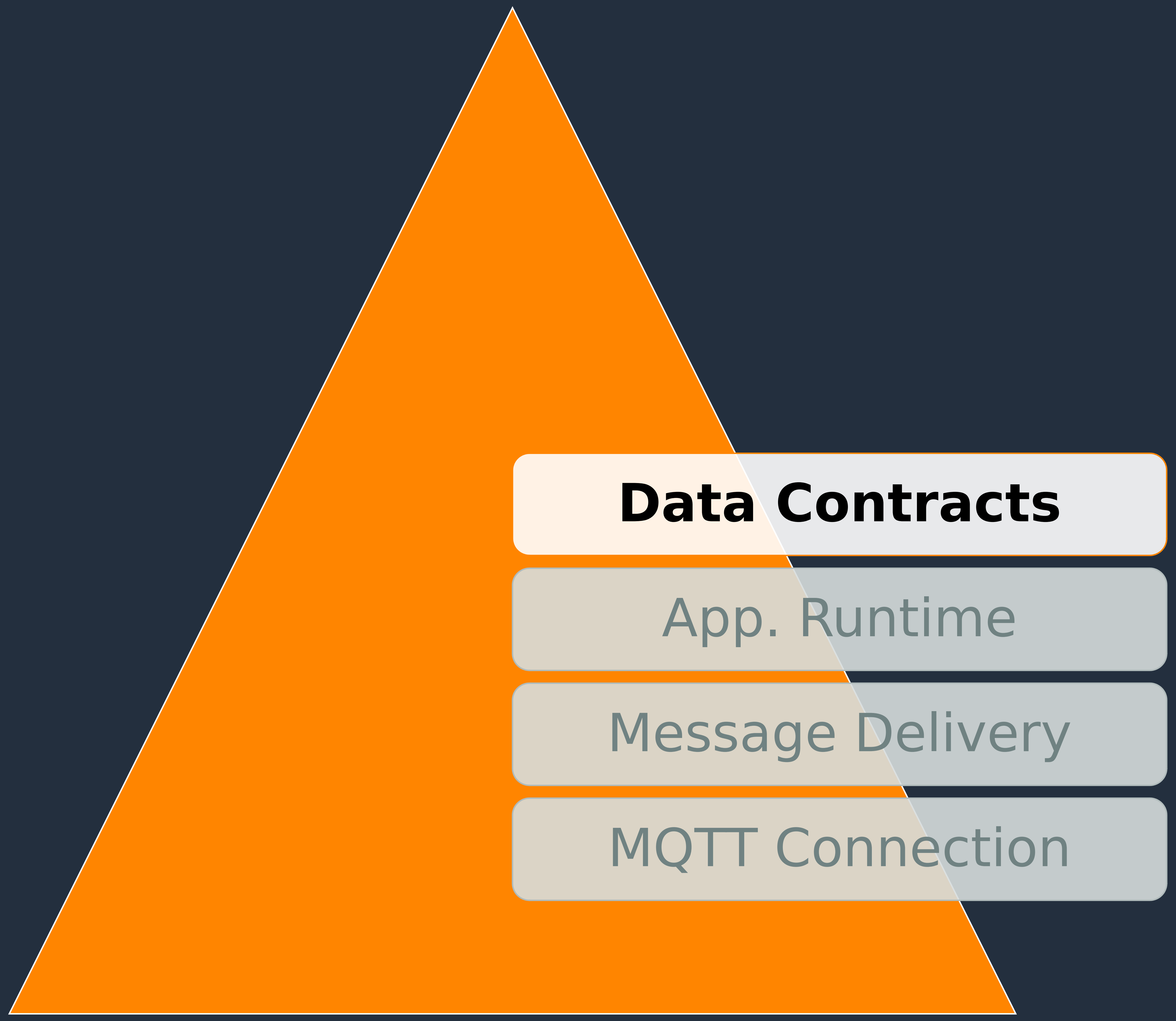


Runtime resilience

Application Level
Ensure your application process recovers and restarts.
Use a process management tool.
Ensure graceful exit handling.
Application logs and metrics.

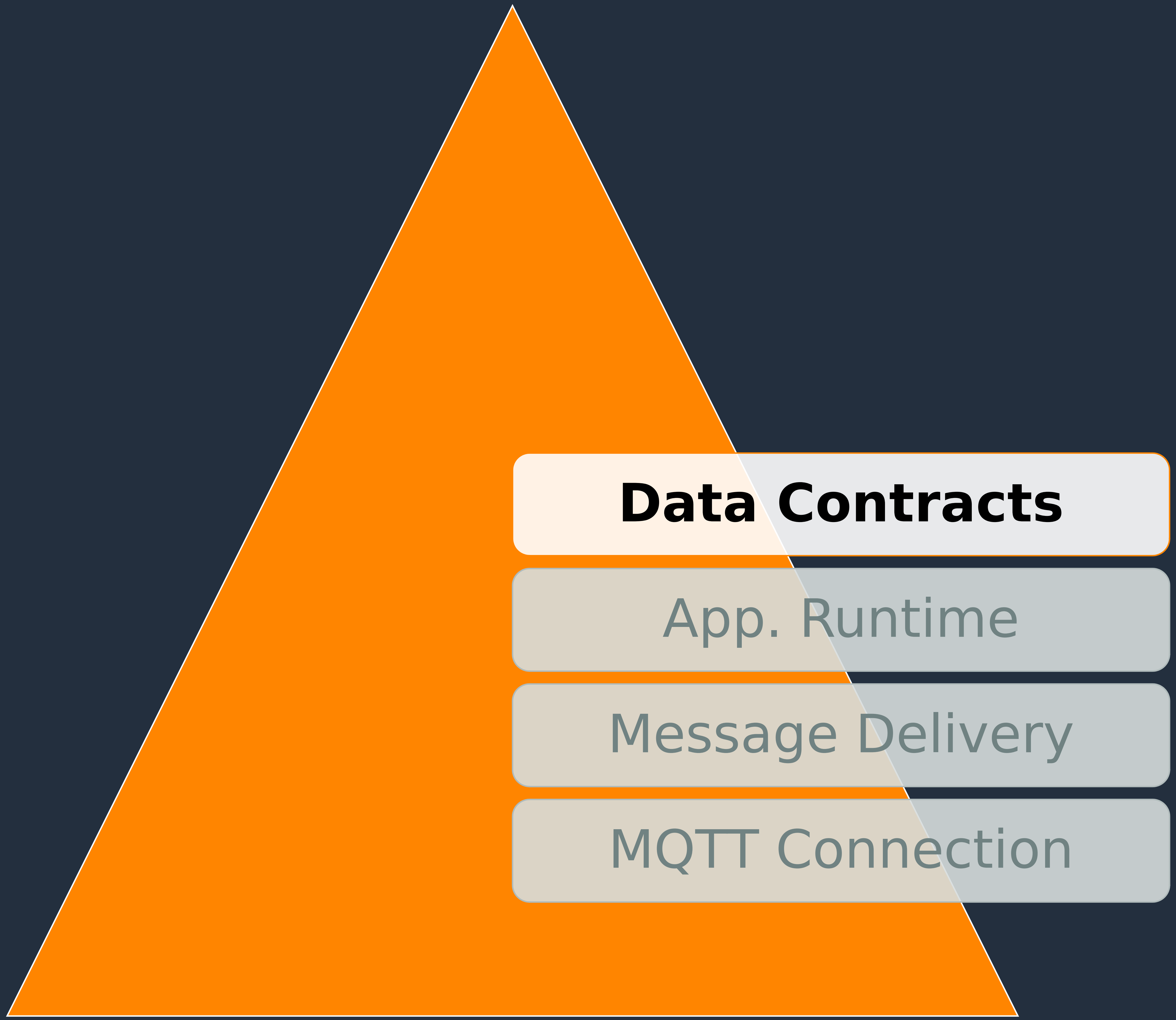


Abstractions and Data Contracts

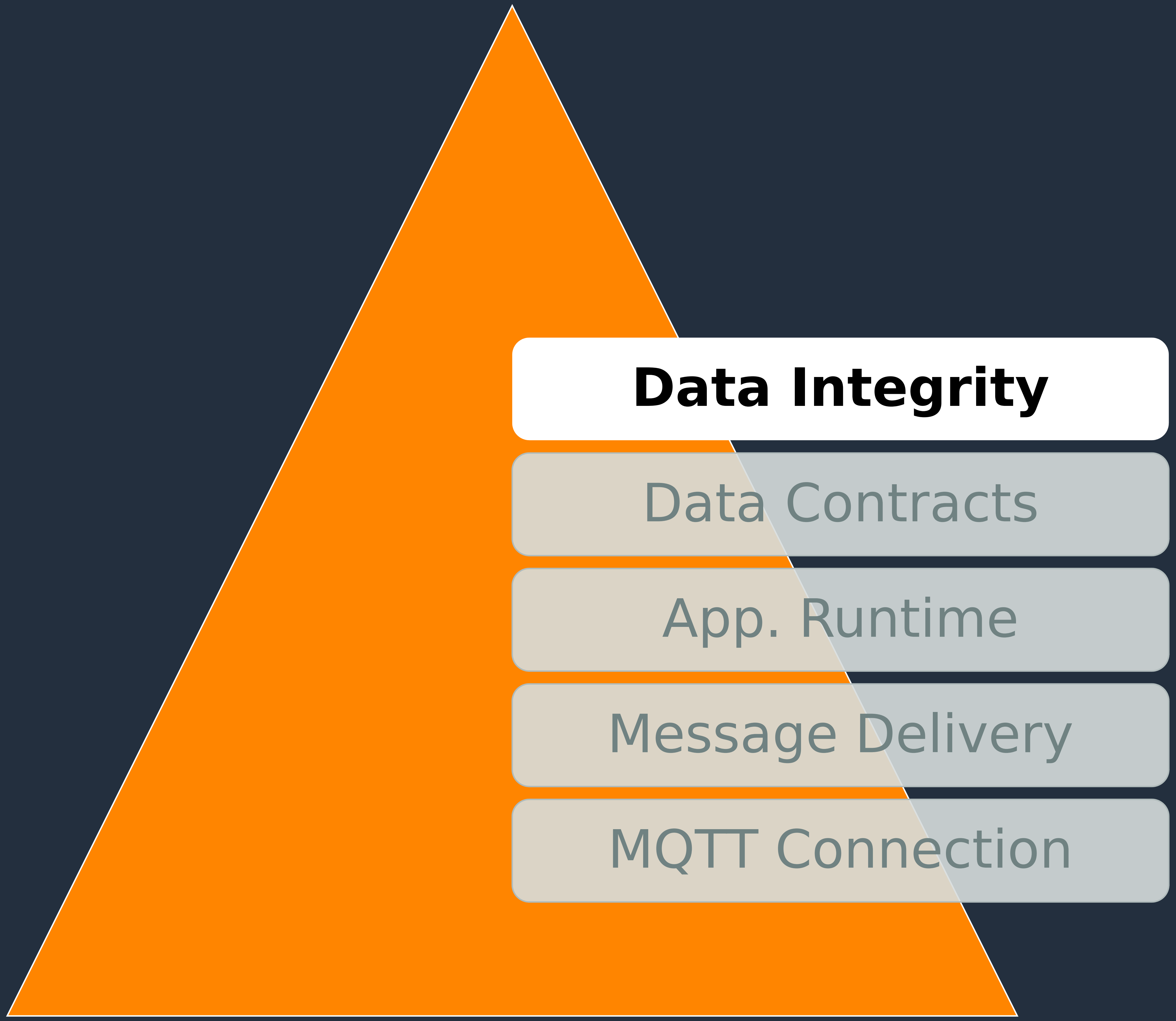


Abstractions and Data Contracts

Application Level
Define your application domain in code: events, metrics, activity, etc.
Enforce data contracts. No breaking changes.
Establish and enforce SLAs.
Send machine raw data, only if it’s what your consumers need.
But be ready to make raw data available during early discovery phases.
Validate.

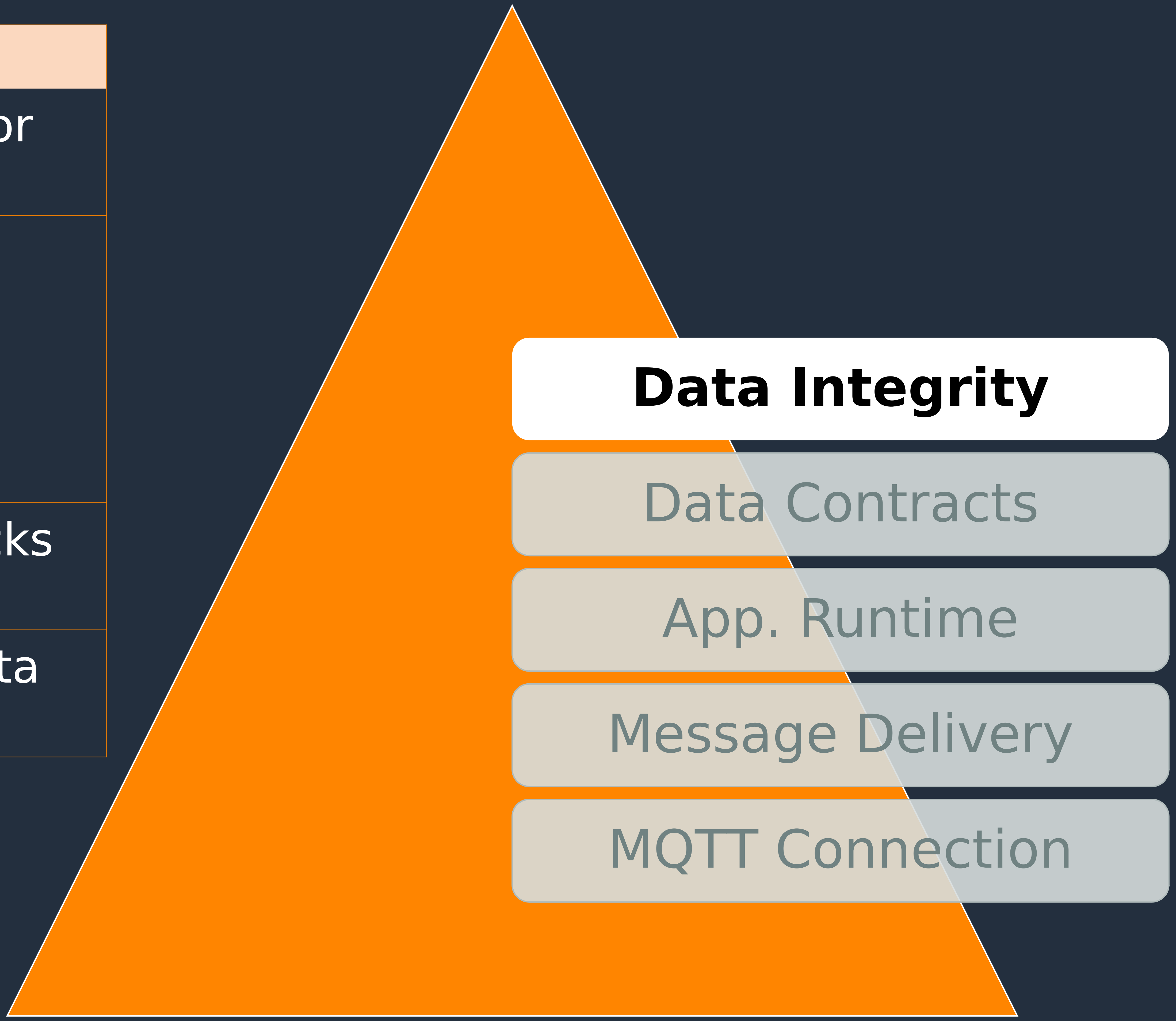


End-to-End Data Integrity

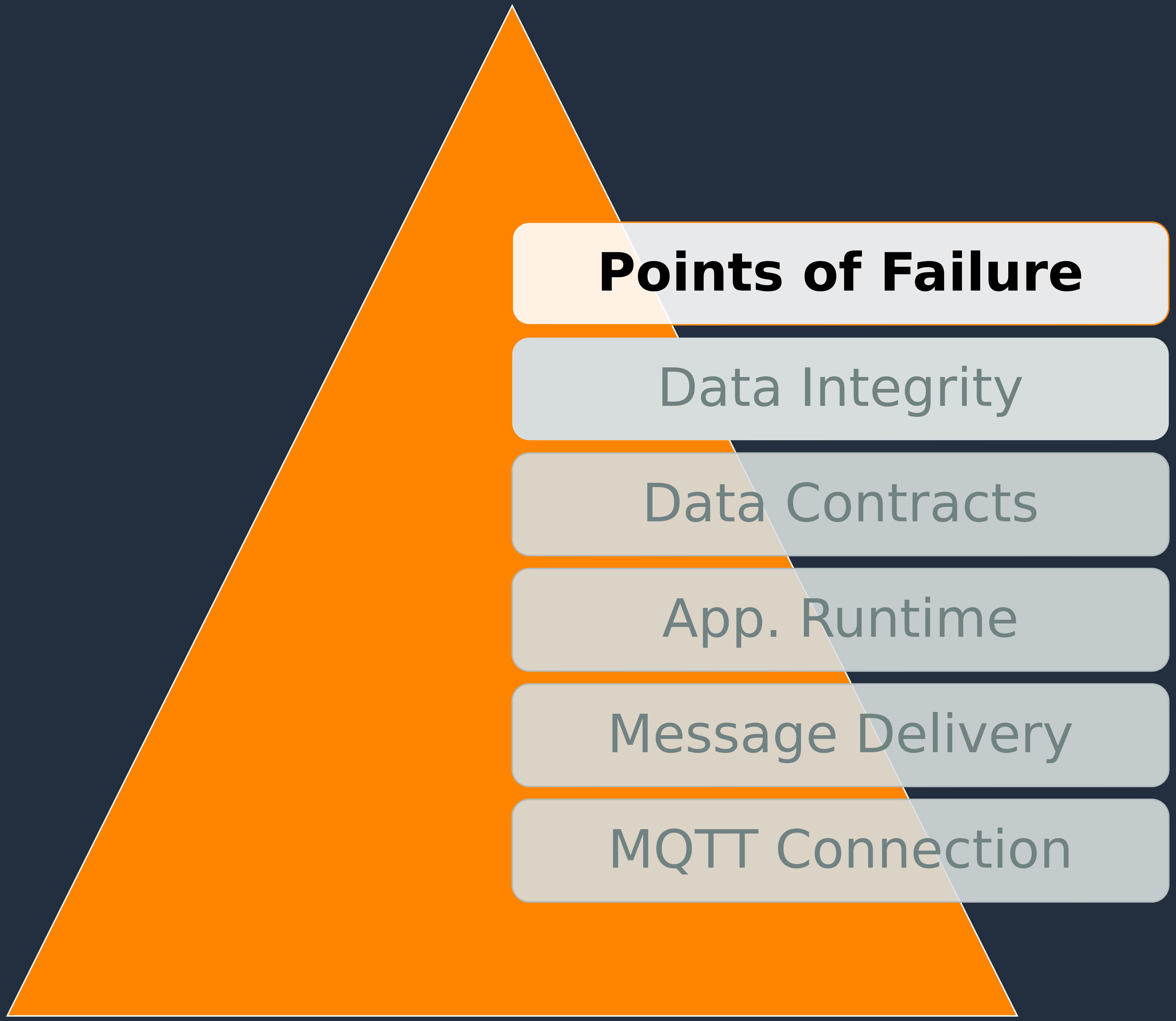


End-to-End Data Integrity

Application Level
Craft what integrity means for your application.
Ensure eventual data consistency: • Checksums, • Timestamps in messages, • Fill in data gaps.
Decouple data integrity checks from your ingestion.
Log, monitor and alert on data integrity issues.



Mitigate all points of failure in your application



Points of Failure

Data Integrity

Data Contracts

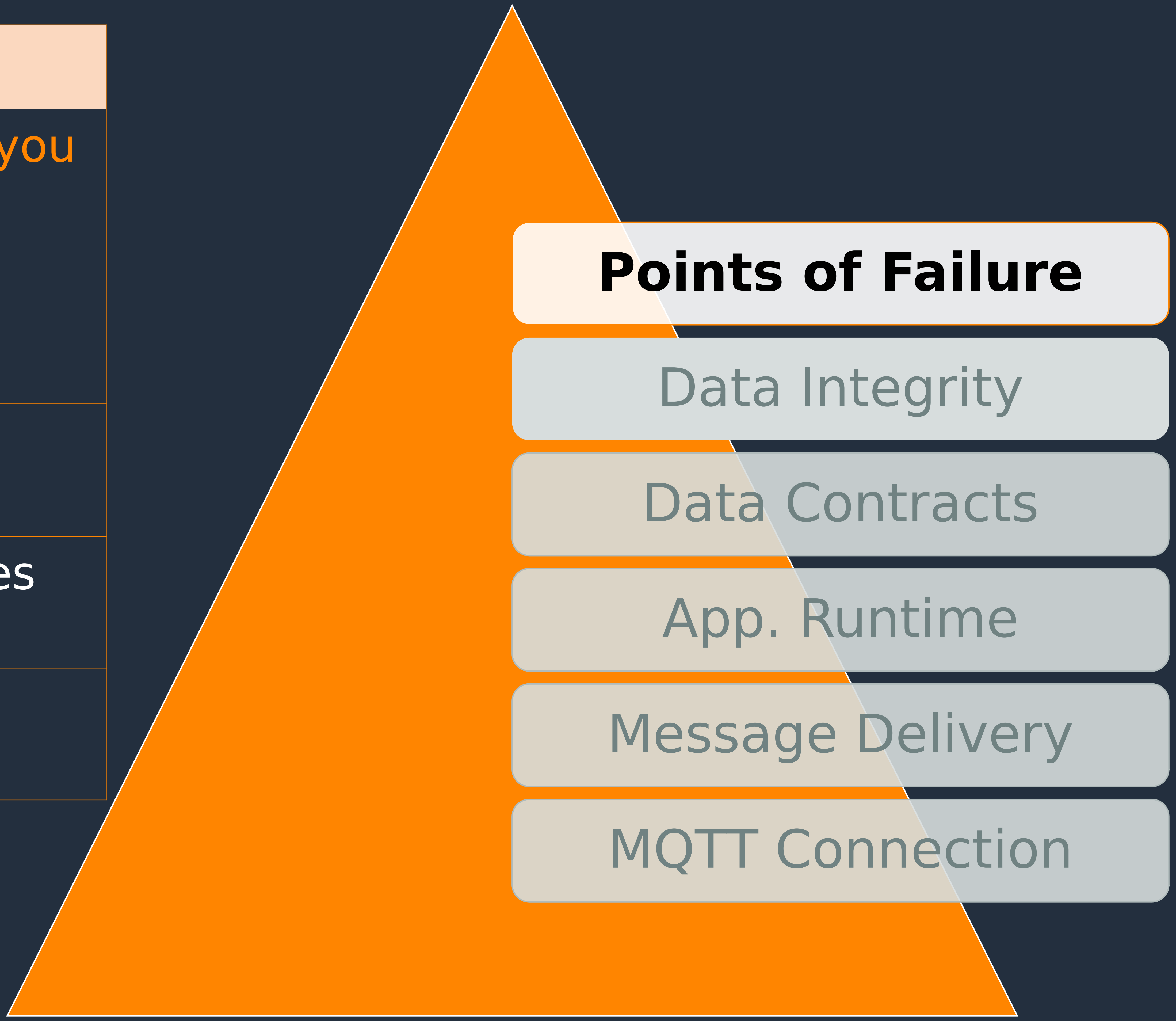
App. Runtime

Message Delivery

MQTT Connection

Mitigate all points of failure in your application

Application Level
Pick the highest abstraction you can accommodate: • Or you are signing up for increased ownership (maintenance).
Answer the question: What happens if it fails?
Manage your points of failures with fallbacks.
Retry with backoff strategy.



Points of Failure

Data Integrity

Data Contracts

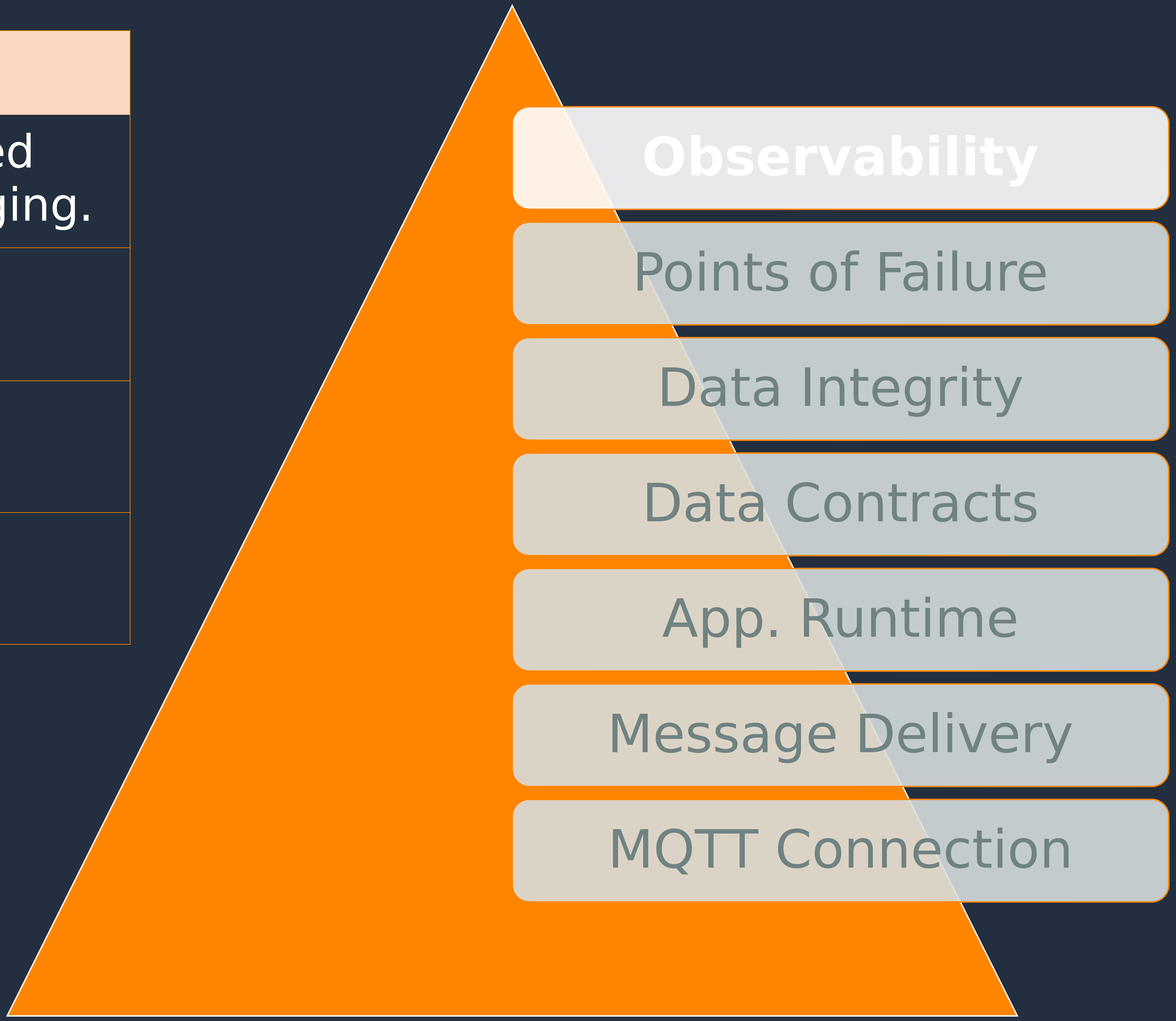
App. Runtime

Message Delivery

MQTT Connection

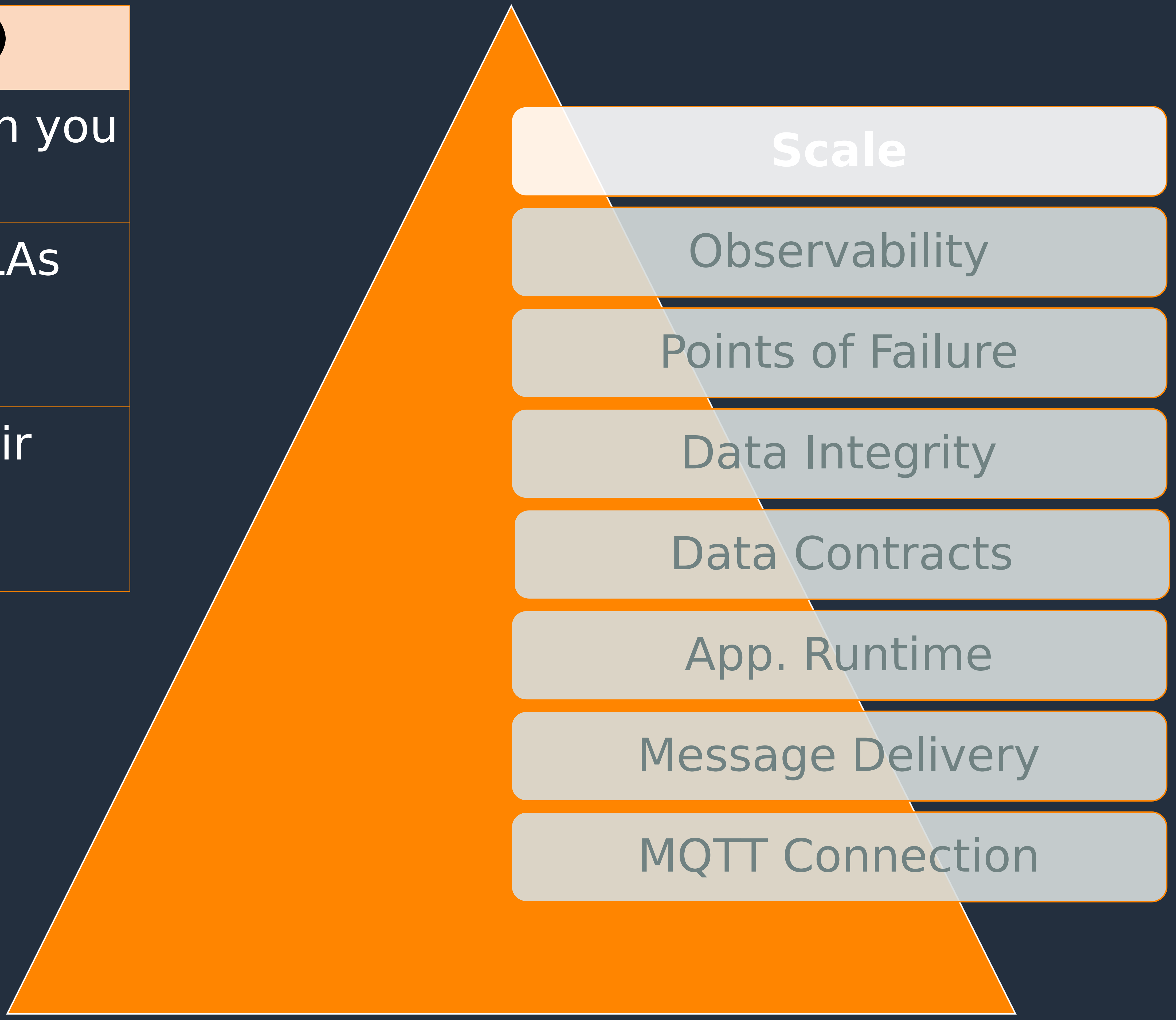
If you can see it, you can fix it! - Observability

Application Level
Standardized and centralized application and service logging.
Metrics.
Tracing.
Compile, analyse, set thresholds, alert/notify.



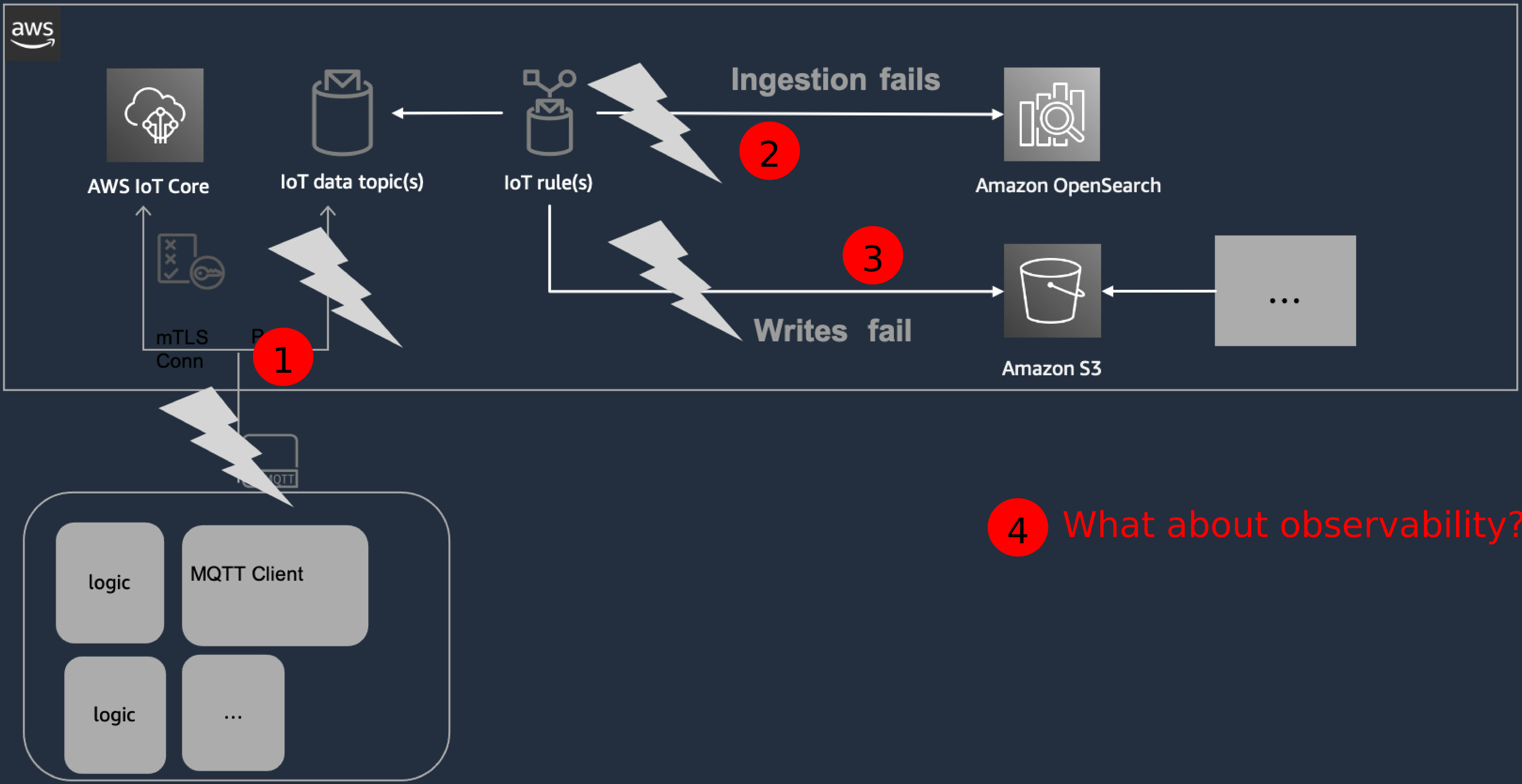
Ready for Scale?

Application Level (Part 1)
Use managed services when you can.
Understand load, service SLAs and if you need to increase service limits.
Devices should un-align their reporting intervals.

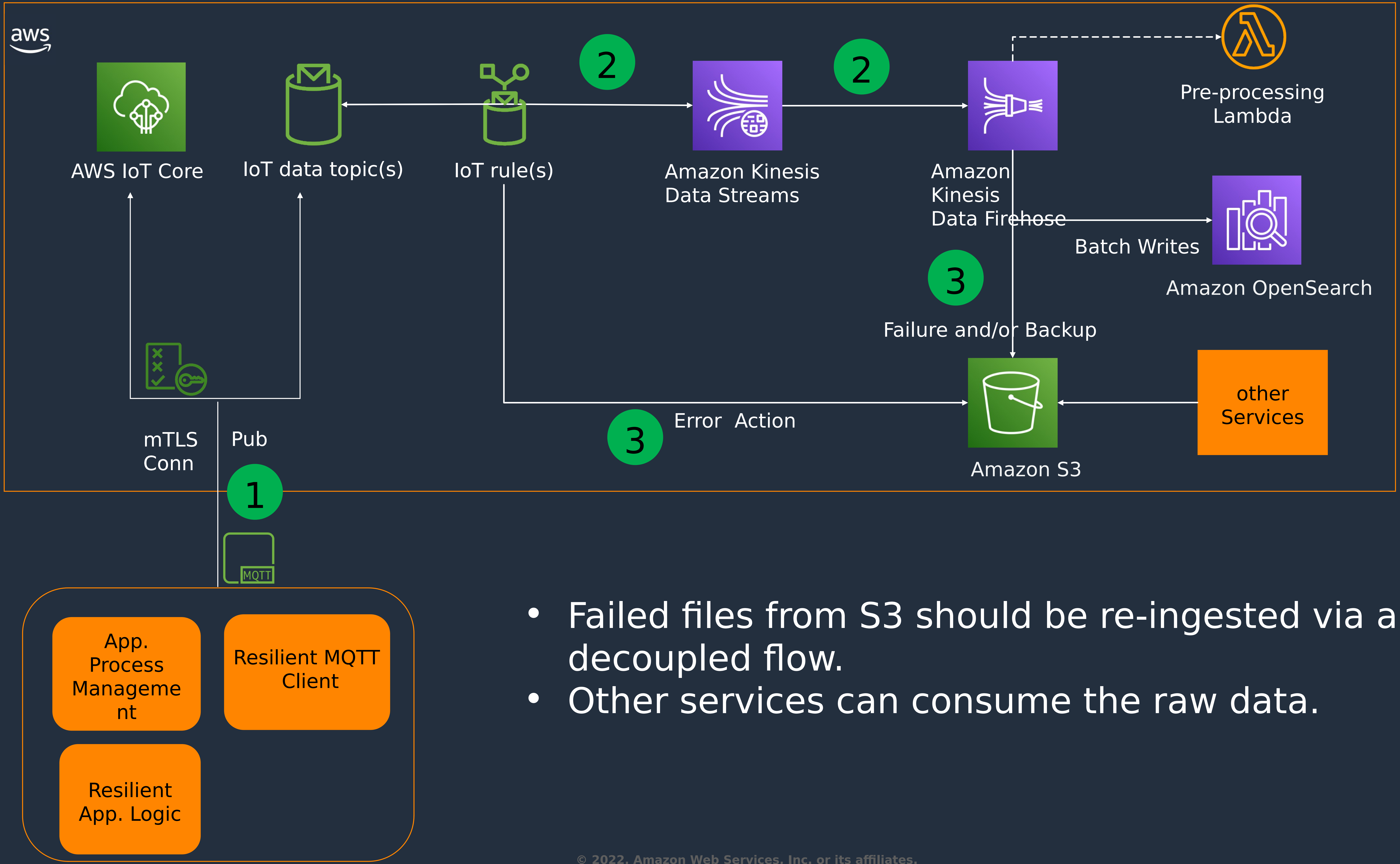


Application Level (Part 2)
Build behaviour for scale in your application: - Scalable reception of data, at different reporting frequencies and volumes, - Scalable routing, - Fan out, - Decouple, - Retries/backoff, - Error Handling, - Observability
Your application can identify what went wrong, and recover.

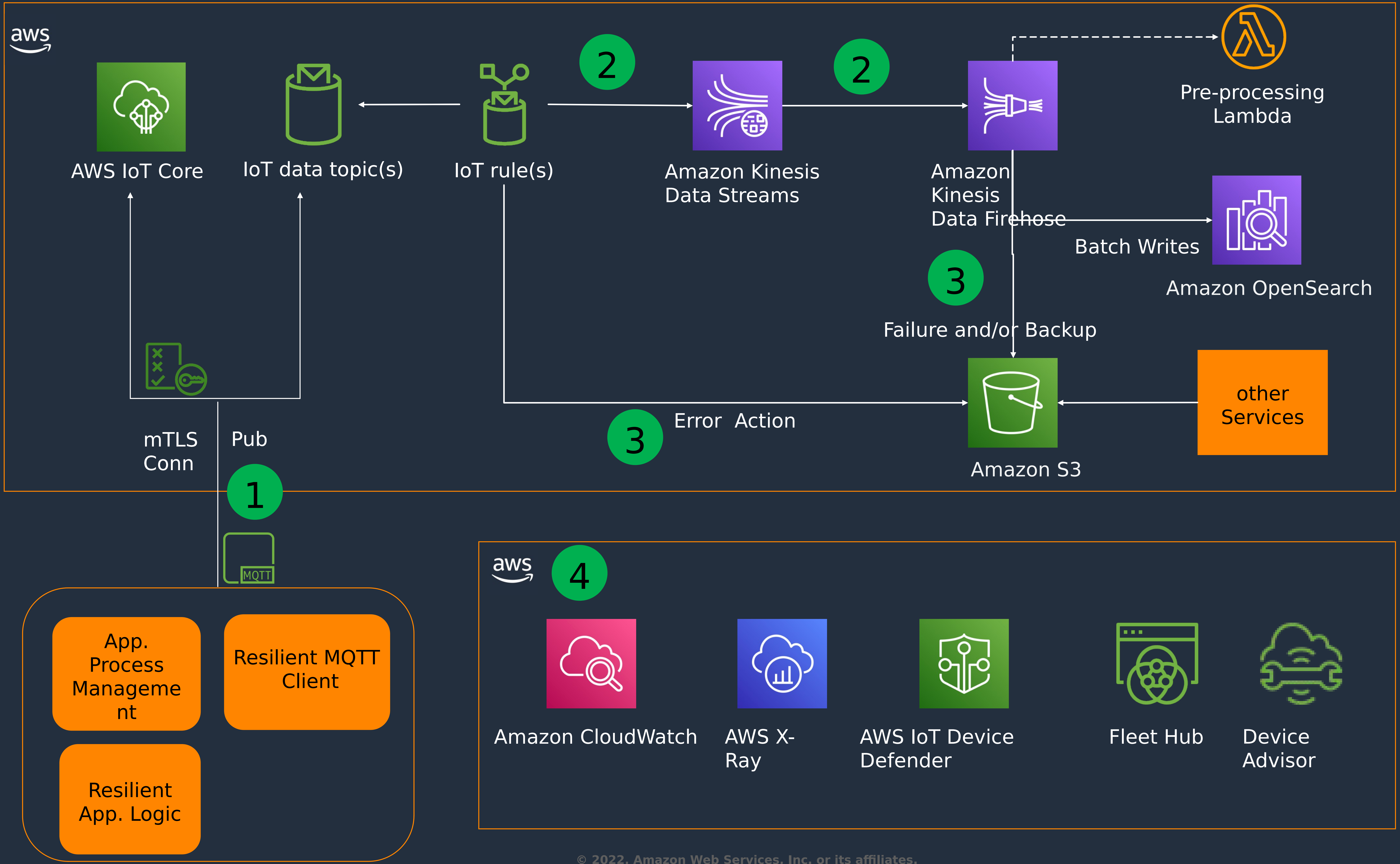
How does this look like end-to-end?



4 What about observability?



- Failed files from S3 should be re-ingested via a decoupled flow.
- Other services can consume the raw data.



AWS IoT Core

IoT data topic(s)

IoT rule(s)

2

Amazon Kinesis Data Streams

2

Amazon Kinesis Data Firehose

Pre-processing Lambda

Batch Writes

Amazon OpenSearch

3

Failure and/or Backup

Error Action

3

Amazon S3

other Services

mTLS Conn

Pub

1

MQTT

App. Process Manageme nt

Resilient MQTT Client

Resilient App. Logic

4

Amazon CloudWatch

AWS X-Ray

AWS IoT Device Defender

Fleet Hub

Device Advisor

Maturity Model for Resilient IoT Applications

Spike/Proof of concept

Hope it works

Know if it works

Mitigate and be sure it works

Ready for Large **Scale**

1 device

100s of devices

Millions of devices

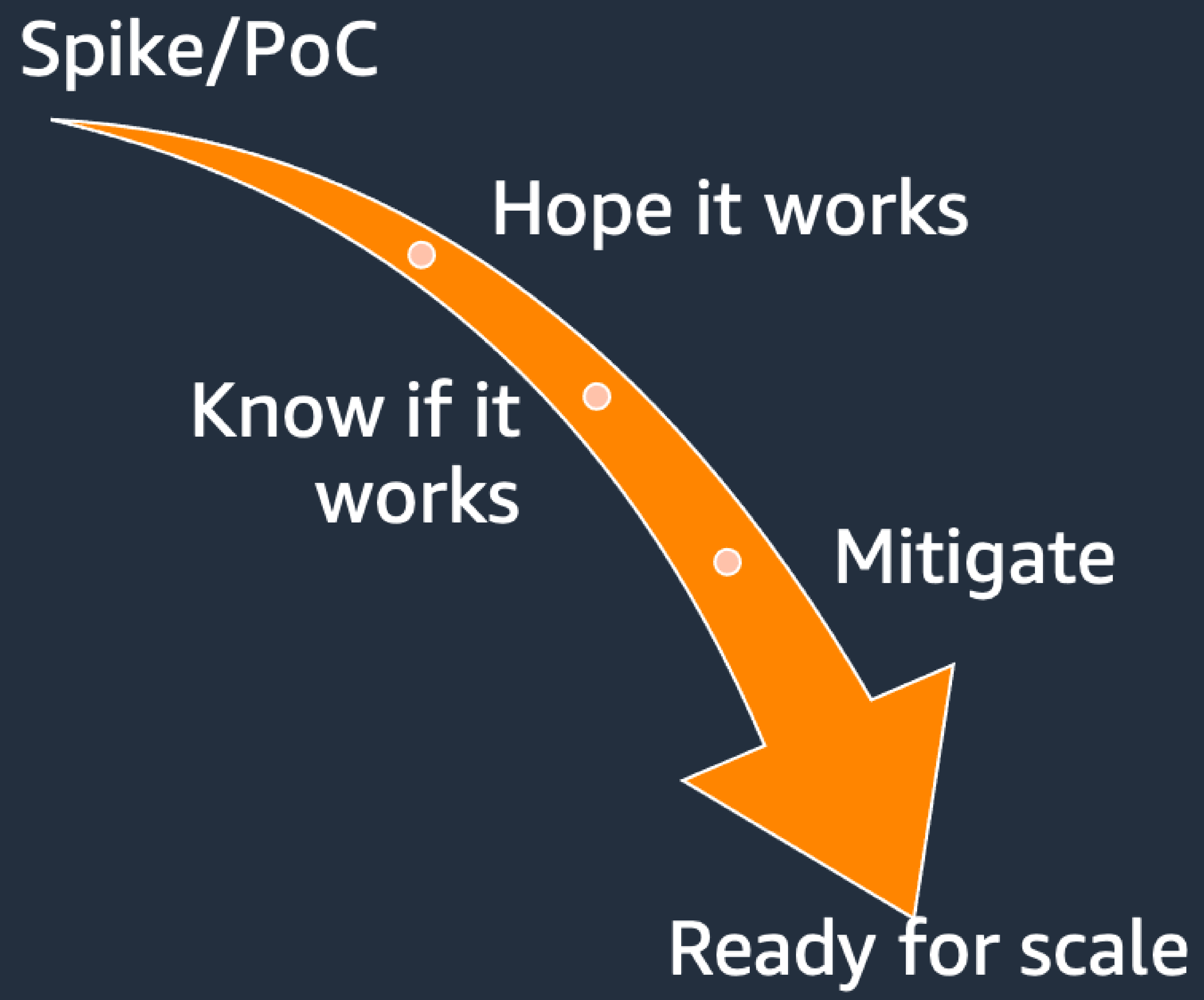
MATURITY MODEL FOR RESILIENT IOT APPLICATIONS

<table>
<tr><th>#</th><th>Phases/Steps</th><th>Edge Application/ One Instance</th><th>Cloud Application – handles many Edge application instances</th></tr>
<tr><td>1</td><td><b>Spike/Proof of concept</b></td><td>• Can connect via MQTT.
• Default client library MQTT configuration can be used.
• Can publish/subscribe via MQTTT.</td><td>• Receives data from an edge application instance.
• Sends that data downstream.</td></tr>
<tr><td>2</td><td><b>Hope it works</b></td><td>• Use MQTT application protocol features for application resilience.
  • What can you just turn on easily? QoS 1, LWT, persistent sessions, retained messages, connection retries with sensible backoff strategies.
• Listen on and handle MQTT connection lifecycle events.</td><td>• Use the highest level of abstraction if possible: for example Cloud managed, serverless technologies, dynamic scaling based on load.</td></tr>
<tr><td rowspan="2">3</td><td rowspan="2"><b>Know if it works</b></td><td colspan="2">•Build logging.
•Log and handle message delivery failures.
•Design for eventual data consistency, and log inconsistencies.
•Use testing tools like the AWS IoT Device Advisor for testing your MQTT connection resilience and security.</td></tr>
<tr><td>•Implement storage at edge.
•Consciously decide how much data you need to store to handle offline-mode.</td><td>•Configure metrics, alarms, tracing.
•Validate reliable and secure connectivity with AWS IoT Core using AWS IoT Core Device Advisor, or similar.</td></tr>
<tr><td>4</td><td><b>Mitigate and be sure it works</b></td><td>•Identify and handle all points of failure.</td><td>•Identify and handle all points of failure (for example: you use an IoT Rule, have an Error Action and write to storage, handle exceptions, tell managed services how to handle errors).
•Ensure your domain data is consistent - sanity checks, check sum algorithms.</td></tr>
<tr><td>5</td><td><b>Ready for large scale</b></td><td></td><td>•Handle millions of requests/messages.
•All calls to external services are surrounded by retries, with reasonable backoff strategies.
•Understand and design taking into account third party system SLAs.
•Watch out for increasing service limits of Cloud services.
•Decouple and fan out.
•Set up monitoring tools: for example Amazon CloudWatch Logs, Metrics, Insights, AWS IoT Device Defender, Fleet Hub.
•Alert the right teams on exceeded thresholds.
•Is your application ready to continuously learn and cope?</td></tr>
</table>

Key Takeaways

Key Takeaways

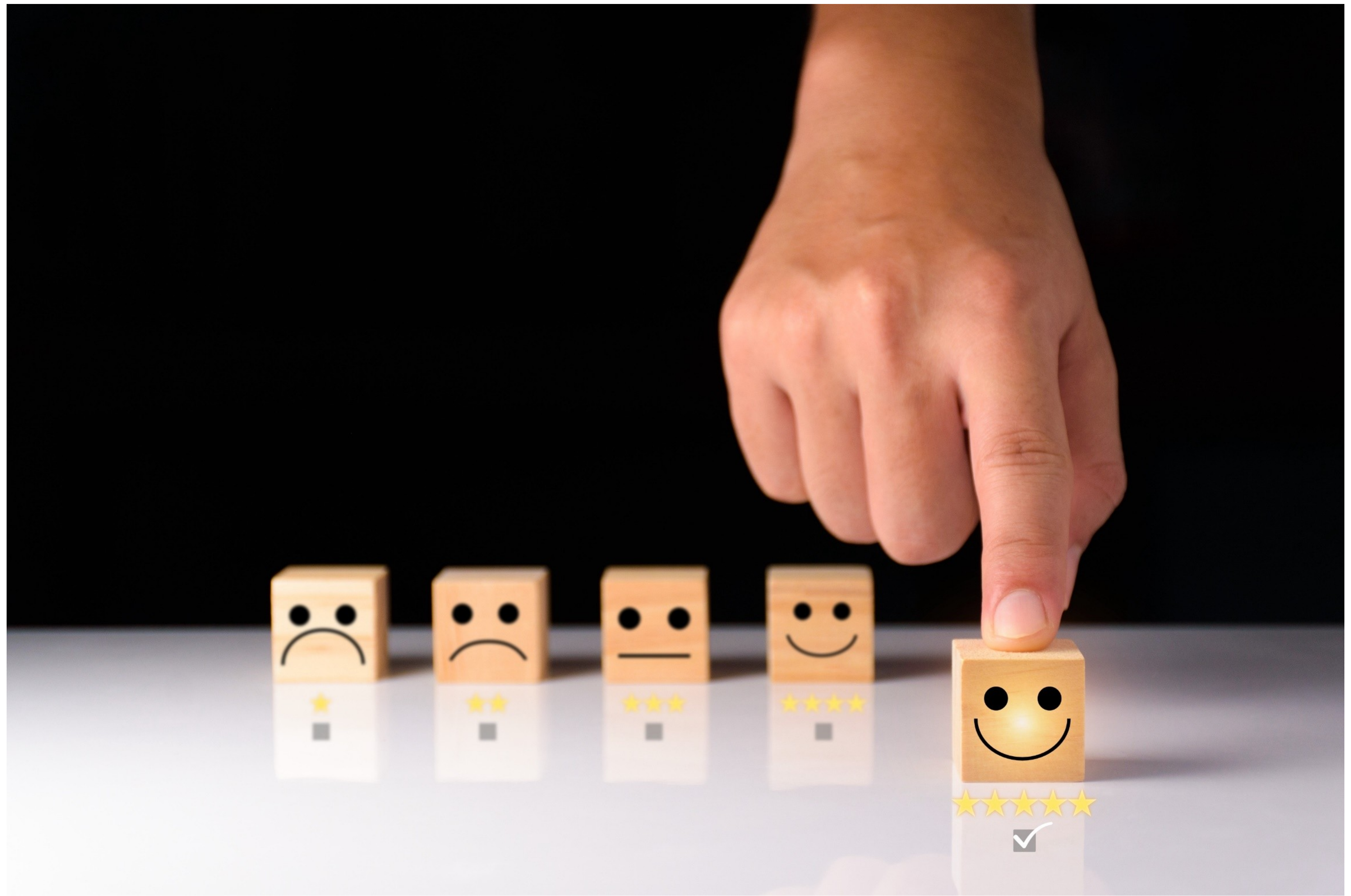
- Accurate insights are not possible with unreliable data.
- Resilience is the mechanism to achieve reliability.
- External & internal factors can cause unreliability in IoT applications.
- Resilience must be built in and we can use a maturity model for resilience.



Resources

- AWS IoT Core Device Advisor: https://docs.aws.amazon.com/iot/latest/developerguide/device-advisor.html
- AWS IoT Device SDK: https://github.com/aws/aws-iot-device-sdk-js-v2
- AWS IoT Greengrass: https://github.com/aws-greengrass
- Blogs/Posts: https://dev.to/iotbuilders
- IoT Dev YouTube: https://youtube.com/@iotbuilders

More Content



Thank you!

Alina Dima

Senior Developer Advocate

AWS IoT

dimaalin@amazon.com

Connect with me